1. Write a C program to generate Fibonacci series of 10 numbers and calculate total number of addition operations and how many times the loop will execute ? total complexity of a program. (4 Times Q-1)

Answer:

#include <stdio.h>

int main() {

int n = 10;

int fib[10];

int addition\_operations = 0; // To count the addition operations

int loop\_executions = 0; // To count the loop executions

// Initializing the first two numbers of the Fibonacci series

fib[0] = 0;

fib[1] = 1;

// Loop to generate Fibonacci series

for (int i = 2; i < n; i++) {

fib[i] = fib[i-1] + fib[i-2];

addition\_operations++;

loop\_executions++;

}

// Print the Fibonacci series

printf("Fibonacci series: ");

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

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

}

printf("\n");

// Print the total number of addition operations and loop executions

printf("Total number of addition operations: %d\n", addition\_operations);

printf("Total number of loop executions: %d\n", loop\_executions + 2); // Include the first two initializations

// The time complexity of the program

printf("Time complexity: O(n)\n");

return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Write a C program to find **minimum of 10 numbers** and calculate time complexity of each statement and total complexity of a program. (1 Time Q-2)

Answer:

#include <stdio.h>

int main() {

int numbers[10] = {34, 7, 23, 32, 5, 62, 3, 78, 1, 9}; // Initial array

int min = numbers[0]; // Initialize the minimum value

int comparisons = 0; // To count the number of comparisons

// Loop to find the minimum value

for (int i = 1; i < 10; i++) { // Loop runs 9 times

comparisons++;

if (numbers[i] < min) { // Each comparison is O(1)

min = numbers[i]; // Assignment is O(1)

}

}

// Print the minimum value

printf("Minimum value: %d\n", min);

printf("Total number of comparisons: %d\n", comparisons);

// The time complexity of the program

printf("Total time complexity: O(n)\n");

return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Write a C program to calculate a factorial of a number and calculate total number of comparison operations and multiplication operations. Also calculate complexity of a program. (2 Times Q-3)

Answer:

#include <stdio.h>

int main() {

int n;

unsigned long long factorial = 1;

int comparisons = 0;

int multiplications = 0;

// Input the number

printf("Enter a number: ");

scanf("%d", &n);

// Calculate the factorial

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

comparisons++; // Each iteration involves one comparison

factorial \*= i;

multiplications++; // Each iteration involves one multiplication

}

// Print the result

printf("Factorial of %d is %llu\n", n, factorial);

printf("Total number of comparisons: %d\n", comparisons);

printf("Total number of multiplications: %d\n", multiplications);

// The time complexity of the program

printf("Total time complexity: O(n)\n");

return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Write a **linear search** C program and do comparison for the worst case. 20

( 8 time -Q-5)

Answer:

#include <stdio.h>

// Function to perform linear search

int linearSearch(int arr[], int n, int x) {

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

if (arr[i] == x) {

return i; // Return the index if element is found

}

}

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

}

int main() {

int arr[] = {10, 23, 45, 70, 11, 15};

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

int x = 70;

int result = linearSearch(arr, n, x);

if (result == -1) {

printf("Element is not present in array\n");

} else {

printf("Element is present at index %d\n", result);

}

return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Write a C program to search for a number 7 in the following sequence of numbers in an array using Linear search : 2, 3, 5, 7, 12, 15, 17 Calculate the total number of comparison operations in the program. How many times will the loop execute ? (4 times repeat)

Answer:

#include <stdio.h>

int main() {

int numbers[] = {2, 3, 5, 7, 12, 15, 17};

int n = sizeof(numbers) / sizeof(numbers[0]);

int target = 7;

int comparisons = 0;

int found = 0; // flag to check if the number is found

// Linear search for the target number

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

comparisons++;

if (numbers[i] == target) {

found = 1;

printf("Number %d found at index %d\n", target, i);

break;

}

}

// If the number is not found in the array

if (!found) {

printf("Number %d not found in the array\n", target);

}

// Print the total number of comparison operations

printf("Total number of comparison operations: %d\n", comparisons);

// Print the total number of loop executions

printf("Total number of loop executions: %d\n", comparisons);

return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Write a C program to find maximum of 10 numbers stored in an array and calculate a total number of comparison operations and complexity of the program

Answer:

#include <stdio.h>

int main() {

int numbers[10] = {34, 7, 23, 32, 5, 62, 3, 78, 1, 9}; // Initial array

int max = numbers[0]; // Initialize the maximum value

int comparisons = 0; // To count the number of comparisons

// Loop to find the maximum value

for (int i = 1; i < 10; i++) {

comparisons++; // Each iteration involves one comparison

if (numbers[i] > max) {

max = numbers[i]; // Update max if a larger number is found

}

}

// Print the maximum value

printf("Maximum value: %d\n", max);

printf("Total number of comparison operations: %d\n", comparisons);

// The time complexity of the program

printf("Total time complexity: O(n)\n");

return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Write a C program to find the maximum of the list 5, 10, 15, 4, 3, 20, 25. How many times will the loop execute ? Calculate the total number of comparison operations in the program and How many times will the loop execute ? (2 Times Q-26)

Answer:

#include <stdio.h>

int main() {

int numbers[] = {5, 10, 15, 4, 3, 20, 25}; // Initial array

int n = sizeof(numbers) / sizeof(numbers[0]);

int max = numbers[0]; // Initialize the maximum value

int comparisons = 0; // To count the number of comparisons

// Loop to find the maximum value

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

comparisons++; // Each iteration involves one comparison

if (numbers[i] > max) {

max = numbers[i]; // Update max if a larger number is found

}

}

// Print the maximum value

printf("Maximum value: %d\n", max);

printf("Total number of comparison operations: %d\n", comparisons);

printf("Total number of loop executions: %d\n", comparisons);

return 0;

}

1. Write a C program to find maximum of 10 numbers stored in an array and calculate a total number of comparison operations and complexity of the program ( Question no 10 - 1 times )

Answer:

#include <stdio.h>

int main() {

int numbers[10] = {34, 7, 23, 32, 5, 62, 3, 78, 1, 9}; // Initial array

int max = numbers[0]; // Initialize the maximum value

int comparisons = 0; // To count the number of comparisons

// Loop to find the maximum value

for (int i = 1; i < 10; i++) {

comparisons++; // Each iteration involves one comparison

if (numbers[i] > max) {

max = numbers[i]; // Update max if a larger number is found

}

}

// Print the maximum value

printf("Maximum value: %d\n", max);

printf("Total number of comparison operations: %d\n", comparisons);

// The time complexity of the program

printf("Total time complexity: O(n)\n");

return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_--

1. Write a C program to accept three numbers from the keyboard and find the maximum number. Calculate the number of comparison operations. (Question no 14 - 1 time )

#include <stdio.h>

int main() {

int num1, num2, num3;

int max;

int comparisons = 0;

// Accept three numbers from the keyboard

printf("Enter first number: ");

scanf("%d", &num1);

printf("Enter second number: ");

scanf("%d", &num2);

printf("Enter third number: ");

scanf("%d", &num3);

// Find the maximum number

max = num1;

comparisons++;

if (num2 > max) {

max = num2;

}

comparisons++;

if (num3 > max) {

max = num3;

}

// Print the maximum number

printf("Maximum number: %d\n", max);

printf("Total number of comparison operations: %d\n", comparisons);

return 0;

}

---------------------------------------------------------------------------------

1. Write a C program to take average of 10 numbers stored in an array. Calculate total number of comparison and addition operations. Also calculate complexity of a program. ( 1 time - Q11)

Answer

#include <stdio.h>

int main() {

int numbers[10] = {34, 7, 23, 32, 5, 62, 3, 78, 1, 9}; // Initial array

int sum = 0; // To store the sum of the numbers

double average; // To store the average

int additions = 0; // To count the number of addition operations

int comparisons = 0; // To count the number of comparison operations

// Loop to calculate the sum of the numbers

for (int i = 0; i < 10; i++) {

sum += numbers[i];

additions++; // Each iteration involves one addition

comparisons++; // Each iteration involves one comparison (the loop condition)

}

// Calculate the average

average = (double)sum / 10;

// Print the average

printf("Average of the numbers: %.2f\n", average);

printf("Total number of addition operations: %d\n", additions);

printf("Total number of comparison operations: %d\n", comparisons);

// The time complexity of the program

printf("Total time complexity: O(n)\n");

return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Write a C Program to Sort the data stored in an array in ascending order using selection sort algorithm and calculate total number of comparison operations. (3 time - Q16)

Answer:

#include <stdio.h>

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

int i, j, min\_idx;

// One by one move boundary of unsorted subarray

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

// Find the minimum element in unsorted array

min\_idx = i;

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

(\*comparisons)++;

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

min\_idx = j;

}

}

// Swap the found minimum element with the first element of the unsorted array

if (min\_idx != i) {

int temp = arr[i];

arr[i] = arr[min\_idx];

arr[min\_idx] = temp;

}

}

}

int main() {

int arr[] = {64, 25, 12, 22, 11};

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

int comparisons = 0;

printf("Original array: ");

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

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

}

printf("\n");

// Perform selection sort

selectionSort(arr, n, &comparisons);

printf("Sorted array: ");

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

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

}

printf("\n");

// Print the total number of comparison operations

printf("Total number of comparison operations: %d\n", comparisons);

return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Write a C program to organize data stored in an array in ascending order and calculate a number of comparison operations for best case and worst case. ( 1 time – Question 13)

Answer:

#include <stdio.h>

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

int i, j, temp;

\*comparisons = 0;

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

// Flag to optimize bubble sort (if no swaps, array is already sorted)

int swapped = 0;

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

(\*comparisons)++;

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

// Swap arr[j] and arr[j+1]

temp = arr[j];

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

arr[j+1] = temp;

swapped = 1;

}

}

// If no two elements were swapped by inner loop, then break

if (swapped == 0)

break;

}

}

int main() {

int arr[] = {64, 25, 12, 22, 11};

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

int comparisons\_best\_case = 0;

int comparisons\_worst\_case = 0;

// Perform bubble sort for best case (already sorted)

bubbleSort(arr, n, &comparisons\_best\_case);

printf("Sorted array (Best Case): ");

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

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

}

printf("\n");

printf("Number of comparison operations (Best Case): %d\n", comparisons\_best\_case);

// Reverse the array for worst case (sorted in descending order)

for (int i = 0; i < n/2; i++) {

int temp = arr[i];

arr[i] = arr[n-i-1];

arr[n-i-1] = temp;

}

// Perform bubble sort for worst case

bubbleSort(arr, n, &comparisons\_worst\_case);

printf("\nSorted array (Worst Case): ");

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

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

}

printf("\n");

printf("Number of comparison operations (Worst Case): %d\n", comparisons\_worst\_case);

return 0;

}

**Bubble Sort Algorithm:**

Bubble sort repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. This process is repeated until the array is sorted.

**Best Case Scenario:**

The best case scenario occurs when the array is already sorted. In this case, bubble sort will make only one pass through the array without any swaps because no swaps are needed.

**Worst Case Scenario:**

The worst case scenario occurs when the array is sorted in reverse order (descending order). In this case, bubble sort will require the maximum number of comparisons and swaps.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Write a C program to accept three numbers from the keyboard and find the maximum number. Calculate the number of comparison operations. ( 1 time – Question 14)

Answer:

#include <stdio.h>

int main() {

int num1, num2, num3;

int max;

int comparisons = 0;

// Accept three numbers from the keyboard

printf("Enter first number: ");

scanf("%d", &num1);

printf("Enter second number: ");

scanf("%d", &num2);

printf("Enter third number: ");

scanf("%d", &num3);

// Find the maximum number

comparisons++; // First comparison between num1 and num2

if (num1 > num2) {

max = num1;

} else {

max = num2;

}

comparisons++; // Second comparison between max and num3

if (num3 > max) {

max = num3;

}

// Print the maximum number

printf("Maximum number: %d\n", max);

printf("Total number of comparison operations: %d\n", comparisons);

return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Write a C program to reverse a string and calculate :

(a) Total number of swap operations

(b) How many times the loop will execute

(Repeat – 5 times Q- 31)

Answer:

#include <stdio.h>

#include <string.h>

void reverseString(char str[], int \*swaps, int \*loop\_count) {

int length = strlen(str);

\*swaps = 0;

\*loop\_count = 0;

for (int i = 0; i < length / 2; i++) {

(\*loop\_count)++; // Counting the number of loop executions

// Swap characters at position i and length-i-1

char temp = str[i];

str[i] = str[length - i - 1];

str[length - i - 1] = temp;

(\*swaps)++; // Counting the number of swap operations

}

}

int main() {

char str[100];

int swaps, loop\_count;

// Input a string from the user

printf("Enter a string: ");

scanf("%s", str);

// Reverse the string

reverseString(str, &swaps, &loop\_count);

// Print the reversed string

printf("Reversed string: %s\n", str);

// Print the number of swap operations

printf("Total number of swap operations: %d\n", swaps);

// Print how many times the loop executed

printf("Number of loop executions: %d\n", loop\_count);

return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Write a C program to Implement a selection sort algorithm to sort an array of 10 integer numbers 25 18 12 14 3 30 35 17 4 11 Calculate the number of exchange and comparison operations. (Repeat 6 times - Q 35 )

Answer:

#include <stdio.h>

void selectionSort(int arr[], int n, int \*exchanges, int \*comparisons) {

\*exchanges = 0;

\*comparisons = 0;

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

int min\_idx = i;

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

(\*comparisons)++;

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

min\_idx = j;

}

}

// Swap arr[i] and arr[min\_idx]

if (min\_idx != i) {

int temp = arr[i];

arr[i] = arr[min\_idx];

arr[min\_idx] = temp;

(\*exchanges)++;

}

}

}

int main() {

int arr[] = {25, 18, 12, 14, 3, 30, 35, 17, 4, 11};

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

int exchanges = 0;

int comparisons = 0;

printf("Original array: ");

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

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

}

printf("\n");

// Perform selection sort

selectionSort(arr, n, &exchanges, &comparisons);

printf("Sorted array: ");

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

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

}

printf("\n");

// Print the number of exchange and comparison operations

printf("Total number of exchange operations: %d\n", exchanges);

printf("Total number of comparison operations: %d\n", comparisons);

return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Write a C program to compute GCD (Greatest Common Divisor). Show running time of each statement and total running time of the program. ( 1 time – Q18)

Answer:

#include <stdio.h>

// Function to compute GCD using Euclidean algorithm

int gcd(int a, int b, int \*running\_time) {

\*running\_time = 0;

int temp;

while (b != 0) {

(\*running\_time)++;

temp = b;

b = a % b;

a = temp;

}

return a;

}

int main() {

int num1, num2;

int result, running\_time;

// Input two numbers from the user

printf("Enter two numbers: ");

scanf("%d %d", &num1, &num2);

// Compute GCD

result = gcd(num1, num2, &running\_time);

// Output the result

printf("GCD of %d and %d is %d\n", num1, num2, result);

printf("Total running time of GCD calculation: %d\n", running\_time);

return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Write a C program to find out both the largest and the smallest integer in an array. Also count how many comparison operations are involved in each. How many times the substraction operation will execute ?

( 5 times Q-19)

Answer:

#include <stdio.h>

#include <limits.h>

void findMinMax(int arr[], int n, int \*largest, int \*smallest, int \*comparisons, int \*subtractions) {

\*largest = INT\_MIN;

\*smallest = INT\_MAX;

\*comparisons = 0;

\*subtractions = 0;

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

(\*comparisons)++; // Increment comparison for checking largest

if (arr[i] > \*largest) {

\*largest = arr[i];

}

(\*comparisons)++; // Increment comparison for checking smallest

if (arr[i] < \*smallest) {

\*smallest = arr[i];

}

// For smallest, add one subtraction operation

\*subtractions += 1;

}

}

int main() {

int arr[] = {25, 18, 12, 14, 3, 30, 35, 17, 4, 11};

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

int largest, smallest, comparisons = 0, subtractions = 0;

// Find largest and smallest

findMinMax(arr, n, &largest, &smallest, &comparisons, &subtractions);

// Output results

printf("Largest integer: %d\n", largest);

printf("Smallest integer: %d\n", smallest);

printf("Total number of comparison operations: %d\n", comparisons);

printf("Total number of subtraction operations for smallest: %d\n", subtractions);

return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Write a C program to organize data stored in an array in descending order and calculate a number of comparison operations for best case and worst case. (1 time Q-21)

Answer:

#include <stdio.h>

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

\*comparisons = 0;

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

int max\_idx = i;

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

(\*comparisons)++;

if (arr[j] > arr[max\_idx]) {

max\_idx = j;

}

}

// Swap arr[i] and arr[max\_idx]

if (max\_idx != i) {

int temp = arr[i];

arr[i] = arr[max\_idx];

arr[max\_idx] = temp;

}

}

}

int main() {

int arr[] = {25, 18, 12, 14, 3, 30, 35, 17, 4, 11};

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

int comparisons\_best\_case = 0, comparisons\_worst\_case = 0;

// Copy the array for best case scenario (already sorted descending)

int arr\_best\_case[n];

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

arr\_best\_case[i] = arr[i];

}

// Copy the array for worst case scenario (sorted ascending)

int arr\_worst\_case[n];

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

arr\_worst\_case[i] = arr[n - 1 - i];

}

// Perform selection sort for best case (already sorted descending)

selectionSortDescending(arr\_best\_case, n, &comparisons\_best\_case);

// Perform selection sort for worst case (sorted ascending)

selectionSortDescending(arr\_worst\_case, n, &comparisons\_worst\_case);

// Output results

printf("Original array: ");

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

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

}

printf("\n");

printf("Array sorted in descending order (Best Case): ");

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

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

}

printf("\n");

printf("Array sorted in descending order (Worst Case): ");

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

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

}

printf("\n");

printf("Number of comparison operations (Best Case): %d\n", comparisons\_best\_case);

printf("Number of comparison operations (Worst Case): %d\n", comparisons\_worst\_case);

return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Write a C program to find the length of a given string. Calculate total number of addition and comparison operations. (2 times – Q22)

Answer:

#include <stdio.h>

int stringLength(char str[], int \*additions, int \*comparisons) {

\*additions = 0;

\*comparisons = 0;

int len = 0;

while (str[len] != '\0') {

(\*comparisons)++;

len++;

(\*additions)++;

}

return len;

}

int main() {

char str[100];

int length, additions, comparisons;

// Input a string from the user

printf("Enter a string: ");

scanf("%s", str);

// Calculate length of the string

length = stringLength(str, &additions, &comparisons);

// Output the results

printf("Length of the string '%s' is %d\n", str, length);

printf("Total number of addition operations: %d\n", additions);

printf("Total number of comparison operations: %d\n", comparisons);

return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Write a C program to calculate the length of a string and calculate the number of times the loop and addition operations will execute. (2 times – Q69)

Answer:

#include <stdio.h>

int stringLength(char str[], int \*loop\_executions, int \*addition\_operations) {

\*loop\_executions = 0;

\*addition\_operations = 0;

int len = 0;

while (str[len] != '\0') {

(\*loop\_executions)++;

len++;

(\*addition\_operations)++;

}

return len;

}

int main() {

char str[100];

int length, loop\_executions, addition\_operations;

// Input a string from the user

printf("Enter a string: ");

scanf("%s", str);

// Calculate length of the string

length = stringLength(str, &loop\_executions, &addition\_operations);

// Output the results

printf("Length of the string '%s' is %d\n", str, length);

printf("Number of loop executions: %d\n", loop\_executions);

printf("Number of addition operations: %d\n", addition\_operations);

return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Write a C program to compute x^n, where both x and n are integer numbers. Calculate the total running time of the program and how many times the loop will execute. (Q- 25 )

Answer:

#include <stdio.h>

long long power(int x, int n, int \*loop\_executions) {

\*loop\_executions = 0;

long long result = 1;

long long base = x;

while (n > 0) {

(\*loop\_executions)++;

if (n % 2 == 1) {

result \*= base;

}

base \*= base;

n /= 2;

}

return result;

}

int main() {

int x, n;

int loop\_executions;

long long result;

// Input values of x and n

printf("Enter the base (x): ");

scanf("%d", &x);

printf("Enter the exponent (n): ");

scanf("%d", &n);

// Calculate x^n using power function

result = power(x, n, &loop\_executions);

// Output the result

printf("%d raised to the power of %d is %lld\n", x, n, result);

printf("Number of loop executions: %d\n", loop\_executions);

return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Write a C program that finds the sum of all the integers in an array. Calculate the total time complexity of a program (Q- 27)

Answer:

#include <stdio.h>

int arraySum(int arr[], int n) {

int sum = 0; // Initialize sum to 0

// Iterate through the array and accumulate sum

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

sum += arr[i]; // Add current element to sum

}

return sum; // Return the computed sum

}

int main() {

int arr[] = {5, 10, 15, 20, 25};

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

// Calculate sum of the array using arraySum function

int sum = arraySum(arr, n);

// Output the result

printf("Sum of all integers in the array is: %d\n", sum);

return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Write a C program to calculate the mean value of a set of six integer numbers stored in an array and calculate how many times the loop and addition operations will execute. The mean of a set of integers is the sum of integers divided by the number of integers in the set. (1 time Q-50)

Answer:

#include <stdio.h>

float calculateMean(int arr[], int n, int \*loop\_executions, int \*addition\_operations) {

\*loop\_executions = 0;

\*addition\_operations = 0;

int sum = 0;

// Calculate sum of array elements

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

(\*loop\_executions)++;

sum += arr[i]; // Add current element to sum

(\*addition\_operations)++;

}

// Calculate mean value

float mean = (float)sum / n;

return mean;

}

int main() {

int arr[] = {5, 10, 15, 20, 25, 30};

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

int loop\_executions, addition\_operations;

// Calculate mean of the array using calculateMean function

float mean = calculateMean(arr, n, &loop\_executions, &addition\_operations);

// Output the result

printf("Mean value of the array elements is: %.2f\n", mean);

printf("Number of loop executions: %d\n", loop\_executions);

printf("Number of addition operations: %d\n", addition\_operations);

return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Write a C program that finds the sum of the square of each integer number stored in an array of size 10. Calculate how many times the assignment operation will execute. (1 time Q-54)

Answer:

#include <stdio.h>

int main() {

int arr[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

int sum\_of\_squares = 0;

int assignment\_operations = 0;

// Calculate sum of squares and count assignment operations

for (int i = 0; i < 10; i++) {

int square = arr[i] \* arr[i];

sum\_of\_squares += square;

assignment\_operations += 2; // One for multiplication and one for addition

}

// Output the result

printf("Sum of squares of array elements is: %d\n", sum\_of\_squares);

printf("Number of assignment operations: %d\n", assignment\_operations);

return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Write a C program to Implement a bubble sort algorithm for sorting the following list of numbers : 27 5 10 8 16 2 37 Calculate how many times the loop will execute. Count the number of comparison operations needed in this program. ( 6 Times Q-41)

Answer:

#include <stdio.h>

void bubbleSort(int arr[], int n, int \*loop\_executions, int \*comparison\_operations) {

\*loop\_executions = 0;

\*comparison\_operations = 0;

int i, j;

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

(\*loop\_executions)++;

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

(\*loop\_executions)++;

(\*comparison\_operations)++;

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

// Swap arr[j] and arr[j+1]

int temp = arr[j];

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

arr[j + 1] = temp;

(\*comparison\_operations)++; // Count the swap operation

}

}

}

}

int main() {

int arr[] = {27, 5, 10, 8, 16, 2, 37};

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

int loop\_executions, comparison\_operations;

// Perform bubble sort on array

bubbleSort(arr, n, &loop\_executions, &comparison\_operations);

// Output sorted array

printf("Sorted array: ");

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

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

}

printf("\n");

// Output the number of loop executions and comparison operations

printf("Number of loop executions: %d\n", loop\_executions);

printf("Number of comparison operations: %d\n", comparison\_operations);

return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Write a C program to check whether a string is a palindrome or not and calculate

(a) the total number of swap operations.

(b) how many times the loop will execute. (Q-31)

Answer:

#include <stdio.h>

#include <string.h>

void reverseString(char str[], int \*loop\_executions, int \*swap\_operations) {

\*loop\_executions = 0;

\*swap\_operations = 0;

int left = 0;

int right = strlen(str) - 1;

while (left < right) {

(\*loop\_executions)++;

if (str[left] != str[right]) {

// Swap characters

char temp = str[left];

str[left] = str[right];

str[right] = temp;

(\*swap\_operations)++;

}

left++;

right--;

}

}

int main() {

char str[100];

int loop\_executions, swap\_operations;

// Input a string from the user

printf("Enter a string: ");

scanf("%s", str);

// Calculate number of swap operations and loop executions

reverseString(str, &loop\_executions, &swap\_operations);

// Check if the original string is the same as the reversed string

int isPalindrome = strcmp(str, strrev(str)) == 0;

// Output the result

if (isPalindrome) {

printf("'%s' is a palindrome.\n", str);

} else {

printf("'%s' is not a palindrome.\n", str);

}

printf("Number of loop executions: %d\n", loop\_executions);

printf("Number of swap operations: %d\n", swap\_operations);

return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Write a C program to Implement a binary search algorithm for searching for an item 16 from the following sorted array A with 10 elements : 2 4 8 12 16 25 30 32 40 50

Calculate the number of comparison operations, divide operation and total time taken to execute a program (3 times Q-33)

Answer:

#include <stdio.h>

int binarySearch(int arr[], int left, int right, int key, int \*comparison\_operations, int \*divide\_operations) {

while (left <= right) {

(\*divide\_operations)++;

int mid = left + (right - left) / 2;

(\*comparison\_operations)++;

// Check if key is present at mid

if (arr[mid] == key) {

return mid; // Key found, return index

}

// If key greater, ignore left half

if (arr[mid] < key) {

left = mid + 1;

}

// If key is smaller, ignore right half

else {

right = mid - 1;

}

}

return -1; // Key not found

}

int main() {

int arr[] = {2, 4, 8, 12, 16, 25, 30, 32, 40, 50};

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

int key = 16;

int comparison\_operations = 0, divide\_operations = 0;

// Perform binary search

int result = binarySearch(arr, 0, n - 1, key, &comparison\_operations, &divide\_operations);

// Output the result

if (result != -1) {

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

} else {

printf("Element %d not found in the array\n", key);

}

printf("Number of comparison operations: %d\n", comparison\_operations);

printf("Number of divide operations: %d\n", divide\_operations);

return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Write a C program to multiply two matrices of order 4 x 4 and calculate total number of comparison, assignment, multiplication and addition operations. (Q-39 4 times)

Answer:

#include <stdio.h>

void multiplyMatrices(int mat1[4][4], int mat2[4][4], int result[4][4],

int \*comparison\_operations, int \*assignment\_operations,

int \*multiplication\_operations, int \*addition\_operations) {

\*comparison\_operations = 0;

\*assignment\_operations = 0;

\*multiplication\_operations = 0;

\*addition\_operations = 0;

// Perform matrix multiplication

for (int i = 0; i < 4; ++i) {

for (int j = 0; j < 4; ++j) {

result[i][j] = 0;

(\*assignment\_operations)++;

for (int k = 0; k < 4; ++k) {

(\*comparison\_operations)++;

(\*multiplication\_operations)++;

(\*addition\_operations)++;

result[i][j] += mat1[i][k] \* mat2[k][j];

(\*assignment\_operations) += 2; // One for multiplication, one for addition

}

}

}

}

int main() {

int mat1[4][4] = {{1, 2, 3, 4},

{5, 6, 7, 8},

{9, 10, 11, 12},

{13, 14, 15, 16}};

int mat2[4][4] = {{17, 18, 19, 20},

{21, 22, 23, 24},

{25, 26, 27, 28},

{29, 30, 31, 32}};

int result[4][4];

int comparison\_operations, assignment\_operations, multiplication\_operations, addition\_operations;

// Multiply matrices and count operations

multiplyMatrices(mat1, mat2, result, &comparison\_operations, &assignment\_operations,

&multiplication\_operations, &addition\_operations);

// Output the result matrix

printf("Resultant Matrix:\n");

for (int i = 0; i < 4; ++i) {

for (int j = 0; j < 4; ++j) {

printf("%d\t", result[i][j]);

}

printf("\n");

}

// Output the number of operations

printf("\nNumber of comparison operations: %d\n", comparison\_operations);

printf("Number of assignment operations: %d\n", assignment\_operations);

printf("Number of multiplication operations: %d\n", multiplication\_operations);

printf("Number of addition operations: %d\n", addition\_operations);

return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Write a C program to Implement insertion sort algorithm to sort an array of 10 integer numbers : 6 20 30 10 15 4 3 17 37 7 Calculate the number of comparison operations executed (5 times Q-53)

Answer:

#include <stdio.h>

void insertionSort(int arr[], int n, int \*comparison\_operations) {

\*comparison\_operations = 0;

int i, key, j;

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

key = arr[i];

j = i - 1;

// Move elements of arr[0..i-1], that are greater than key, to one position ahead

// of their current position

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

(\*comparison\_operations)++;

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

j = j - 1;

}

arr[j + 1] = key;

}

}

int main() {

int arr[] = {6, 20, 30, 10, 15, 4, 3, 17, 37, 7};

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

int comparison\_operations;

// Perform insertion sort and count comparison operations

insertionSort(arr, n, &comparison\_operations);

// Output the sorted array

printf("Sorted array: ");

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

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

}

printf("\n");

// Output the number of comparison operations

printf("Number of comparison operations: %d\n", comparison\_operations);

return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Write a C program to evaluate a polynomial using Horner's rule and count how many times the loop will execute. (Q-73)

Answer:

#include <stdio.h>

// Function to evaluate a polynomial using Horner's rule

int evaluatePolynomial(int poly[], int n, int x, int \*loop\_executions) {

int result = poly[0]; // Initialize result with coefficient of x^n

\*loop\_executions = 0; // Initialize loop execution count

// Evaluate the polynomial using Horner's rule

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

\*loop\_executions++;

result = result \* x + poly[i];

}

return result;

}

int main() {

int poly[] = {3, 2, -5, 4}; // Example polynomial: 3x^3 + 2x^2 - 5x + 4

int n = sizeof(poly) / sizeof(poly[0]) - 1; // Degree of the polynomial

int x = 2; // Value of x for evaluation

int loop\_executions;

// Evaluate polynomial using Horner's rule and count loop executions

int result = evaluatePolynomial(poly, n, x, &loop\_executions);

// Output the result and number of loop executions

printf("Result of polynomial evaluation: %d\n", result);

printf("Number of loop executions: %d\n", loop\_executions);

return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Write a C program to copy a string to another string. Count how many times the assignment operations and loop will execute. ( Q-79)

Answer:

#include <stdio.h>

void copyString(char \*source, char \*destination, int \*assignment\_operations, int \*loop\_executions) {

\*assignment\_operations = 0;

\*loop\_executions = 0;

// Copy characters from source to destination until null character '\0' is encountered

while (\*source) {

\*assignment\_operations += 1; // For copying character

\*loop\_executions += 1; // Count loop execution

\*destination = \*source; // Copy character from source to destination

source++; // Move source pointer to next character

destination++; // Move destination pointer to next character

}

\*assignment\_operations += 1; // For copying null character '\0'

\*destination = '\0'; // Append null character to end the destination string

}

int main() {

char source[] = "Hello, world!"; // Example source string

char destination[50]; // Destination string with sufficient size

int assignment\_operations, loop\_executions;

// Copy source string to destination string and count operations

copyString(source, destination, &assignment\_operations, &loop\_executions);

// Output the copied string and number of operations

printf("Source string: %s\n", source);

printf("Copied string: %s\n", destination);

printf("Number of assignment operations: %d\n", assignment\_operations);

printf("Number of loop executions: %d\n", loop\_executions);

return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Write a C program to compute a nby left to right binary exponentiation method and , count how many times the loops will execute ? (Q-83)

Answer:

#include <stdio.h>

long long power(int a, int n, int \*loop\_executions) {

long long result = 1;

\*loop\_executions = 0;

while (n > 0) {

(\*loop\_executions)++;

if (n % 2 == 1) { // if n is odd

result \*= a;

}

a \*= a; // square the base

n /= 2; // divide the exponent by 2

}

return result;

}

int main() {

int a = 3; // base

int n = 10; // exponent

int loop\_executions;

// Compute a^n using left-to-right binary exponentiation

long long result = power(a, n, &loop\_executions);

// Output the result and number of loop executions

printf("%d^%d = %lld\n", a, n, result);

printf("Number of loop executions: %d\n", loop\_executions);

return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Write a C program that finds the location of the first odd number in an array of integer numbers. Calculate the number of times the loop and the arithmetic operation to test whether a number is odd or even will execute.

Answer:

#include <stdio.h>

int findFirstOdd(int arr[], int size, int \*loop\_executions, int \*arithmetic\_operations) {

\*loop\_executions = 0;

\*arithmetic\_operations = 0;

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

(\*loop\_executions)++; // Counting loop executions

if (arr[i] % 2 != 0) { // Checking if the number is odd

(\*arithmetic\_operations)++; // Counting arithmetic operation for odd/even check

return i; // Return index of first odd number found

}

(\*arithmetic\_operations)++; // Counting arithmetic operation for modulo operation

}

return -1; // Return -1 if no odd number is found

}

int main() {

int arr[] = {2, 4, 6, 8, 10, 11, 12, 14, 16}; // Example array

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

int loop\_executions, arithmetic\_operations;

// Find location of first odd number in the array and count operations

int index = findFirstOdd(arr, size, &loop\_executions, &arithmetic\_operations);

// Output the result and number of operations

if (index != -1) {

printf("First odd number found at index %d\n", index);

} else {

printf("No odd number found in the array\n");

}

printf("Number of loop executions: %d\n", loop\_executions);

printf("Number of arithmetic operations: %d\n", arithmetic\_operations);

return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Write a C program that finds the location of the last even integer number in an array. Calculate the number of time's. the loop and arithmetic operation to test whether the number is an even integer number or n

Answer:

#include <stdio.h>

int findLastEven(int arr[], int size, int \*loop\_executions, int \*arithmetic\_operations) {

\*loop\_executions = 0;

\*arithmetic\_operations = 0;

int lastIndex = -1;

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

(\*loop\_executions)++; // Counting loop executions

if (arr[i] % 2 == 0) { // Checking if the number is even

lastIndex = i; // Update the index of the last even number found

(\*arithmetic\_operations)++; // Counting arithmetic operation for even check

}

(\*arithmetic\_operations)++; // Counting arithmetic operation for modulo operation

}

return lastIndex; // Return index of the last even number found (-1 if none found)

}

int main() {

int arr[] = {1, 3, 5, 7, 9, 10, 12, 14, 16}; // Example array

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

int loop\_executions, arithmetic\_operations;

// Find location of last even number in the array and count operations

int index = findLastEven(arr, size, &loop\_executions, &arithmetic\_operations);

// Output the result and number of operations

if (index != -1) {

printf("Last even number found at index %d\n", index);

} else {

printf("No even number found in the array\n");

}

printf("Number of loop executions: %d\n", loop\_executions);

printf("Number of arithmetic operations: %d\n", arithmetic\_operations);

return 0;

}

===========================================================================

1. Quick Sort Algorithm

Answer:

#include <stdio.h>

void quickSort(int arr[], int low, int high, int \*comparisons, int \*divisions);

int partition(int arr[], int low, int high, int \*comparisons, int \*divisions);

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

int main() {

int arr[] = {25, 18, 12, 14, 3, 30, 35, 17, 4, 11}; // Example array

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

int comparisons = 0, divisions = 0;

// Perform Quick Sort

quickSort(arr, 0, n - 1, &comparisons, &divisions);

// 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 divisions

printf("Number of comparisons: %d\n", comparisons);

printf("Number of divisions: %d\n", divisions);

return 0;

}

void quickSort(int arr[], int low, int high, int \*comparisons, int \*divisions) {

if (low < high) {

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

quickSort(arr, low, pi - 1, comparisons, divisions); // Recursively sort left subarray

quickSort(arr, pi + 1, high, comparisons, divisions); // Recursively sort right subarray

}

}

int partition(int arr[], int low, int high, int \*comparisons, int \*divisions) {

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

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

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

(\*comparisons)++; // Counting comparisons

if (arr[j] < pivot) {

i++;

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

}

}

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

(\*divisions)++; // Counting divisions (partitions)

return (i + 1);

}

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

int t = \*a;

\*a = \*b;

\*b = t;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Merge sort algorithm

Answer:

#include <stdio.h>

void mergeSort(int arr[], int l, int r, int \*comparisons, int \*divisions);

void merge(int arr[], int l, int m, int r, int \*comparisons);

int main() {

int arr[] = {25, 18, 12, 14, 3, 30, 35, 17, 4, 11}; // Example array

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

int comparisons = 0, divisions = 0;

// Perform Merge Sort

mergeSort(arr, 0, n - 1, &comparisons, &divisions);

// 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 divisions

printf("Number of comparisons: %d\n", comparisons);

printf("Number of divisions: %d\n", divisions);

return 0;

}

void mergeSort(int arr[], int l, int r, int \*comparisons, int \*divisions) {

if (l < r) {

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

// Recursively sort first and second halves

mergeSort(arr, l, m, comparisons, divisions);

mergeSort(arr, m + 1, r, comparisons, divisions);

// Merge the sorted halves

merge(arr, l, m, r, comparisons);

(\*divisions)++; // Counting division (split) operations

}

}

void merge(int arr[], int l, int m, int r, int \*comparisons) {

int n1 = m - l + 1;

int n2 = r - m;

// Create temporary arrays

int L[n1], R[n2];

// Copy data to temporary arrays L[] and R[]

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

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

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

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

// Merge the temporary arrays back into arr[l..r]

int i = 0, j = 0, k = l;

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

(\*comparisons)++; // Counting comparisons

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++;

}

}

==========================================================

1. For the following program, find the time complexity 20

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

Printf(“First Loop”) (Question No 4)

Answer:

**Analysis**

1. **Initialization:** i = 0
2. **Condition Check:** i < n
3. **Body Execution:** printf("First Loop")
4. **Increment:** i++

**Steps in the Loop:**

* The loop runs from i = 0 to i = n - 1.
* The condition check i < n happens n+1 times (including the final check when i equals n).
* The printf statement executes n times.
* The increment i++ executes n times.

**Time Complexity:**

* The time complexity is primarily determined by the number of times the printf statement is executed.
* Each operation inside the loop (condition check, printf, and increment) is performed n times.
* Therefore, the total time complexity of the loop is O(n).

**Conclusion:**

The time complexity of the given program is O(n).

This linear time complexity indicates that the number of operations grows linearly with the size of the input n.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. For the following program fragment find the complexity

for (i = 0; i< n; i+ +); (Question No 6)

Answer:

**Analysis**

1. **Initialization:** i = 0
2. **Condition Check:** i < n
3. **Increment:** i++
4. **Empty Statement:** The loop body is empty (indicated by the semicolon ;).

**Steps in the Loop:**

* The loop runs from i = 0 to i = n - 1.
* The condition check i < n happens n+1 times (including the final check when i equals n).
* The increment i++ executes n times.
* Since the loop body is empty, there are no operations performed inside the loop.

**Time Complexity:**

* The time complexity is primarily determined by the number of times the condition check and the increment operations are executed.
* Both the condition check and the increment operation are performed n times.

**Conclusion:**

Even though there are no operations inside the loop body, the loop itself still performs a constant amount of work per iteration due to the condition check and increment operations.

Therefore, the time complexity of the given program fragment is O(n).

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Find the time complexity of the following program fragment 20

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

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

for (k = 0; k< n; k+ +);

printf('inner looping statement); (Question No 8)

Answer:

**Analysis**

1. **First Loop:**
   * Initialization: i = 0
   * Condition Check: i < n
   * Increment: i++
   * The loop body is empty (indicated by the semicolon ;).

This loop runs nnn times.

1. **Second Loop:**
   * Initialization: j = 0
   * Condition Check: j < n
   * Increment: j++
   * The loop body is empty (indicated by the semicolon ;).

This loop runs nnn times.

1. **Third Loop:**
   * Initialization: k = 0
   * Condition Check: k < n
   * Increment: k++
   * The loop body is empty (indicated by the semicolon ;).

This loop runs nnn times.

1. **Print Statement:**
   * The printf statement executes once after all the loops are finished.

**Time Complexity:**

* Each of the three loops runs independently of the others.
* Each loop runs nnn times, but they are not nested.
* Therefore, the time complexity for each loop is O(n) .

Since the loops are sequential, we sum their time complexities:

O(n)+O(n)+O(n)=3O(n)

However, in Big-O notation, we drop constant factors:

3O(n)=O(n)

The printf statement executes only once, which is O(1).

**Conclusion:**

The overall time complexity of the given program fragment is:

O(n)

This indicates that the number of operations grows linearly with the size of the input n.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. For the following program, find the time complexity.

For(i=0; i<n; i++)

For(j=0; j<n; j++)

print f ("first + second loop"); (Question No 12)

Answer:

**Analysis**

1. **Outer Loop (i loop):**
   * Initialization: i = 0
   * Condition Check: i < n
   * Increment: i++
   * Runs nnn times.
2. **Inner Loop (j loop):**
   * Initialization: j = 0
   * Condition Check: j < n
   * Increment: j++
   * Runs nnn times for each iteration of the outer loop.
3. **Print Statement:**
   * The printf statement executes once per iteration of the inner loop.

**Steps in the Loop:**

* The outer loop runs nnn times.
* For each iteration of the outer loop, the inner loop also runs nnn times.
* Therefore, the total number of executions of the printf statement is n×n=n^2.

**Time Complexity:**

* The outer loop has O(n) complexity.
* The inner loop, nested within the outer loop, also has O(n) complexity.

When loops are nested, their complexities multiply:

O(n)×O(n)= O(n^2)

**Conclusion:**

The time complexity of the given program is:

O(n^2)

This quadratic time complexity indicates that the number of operations grows proportionally to the square of the input size n.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. For the following program, find the time complexity.

For(i=0; i<n; i++)

For(j=0; j<n; j++)

For(k=0; k<n; k++)

print f ("first + second +third loop"); ( Q-20)

Answer:

**Analysis**

1. **Outer Loop (i loop):**
   * Initialization: i = 0
   * Condition Check: i < n
   * Increment: i++
   * Runs nnn times.
2. **Middle Loop (j loop):**
   * Initialization: j = 0
   * Condition Check: j < n
   * Increment: j++
   * Runs nnn times for each iteration of the outer loop.
3. **Inner Loop (k loop):**
   * Initialization: k = 0
   * Condition Check: k < n
   * Increment: k++
   * Runs nnn times for each iteration of the middle loop.
4. **Print Statement:**
   * The printf statement executes once per iteration of the inner loop.

**Steps in the Loop:**

* The outer loop runs n times.
* For each iteration of the outer loop, the middle loop runs n times.
* For each iteration of the middle loop, the inner loop runs n times.
* Therefore, the total number of executions of the printf statement is n×n×n=n^3.

**Time Complexity:**

* The outer loop has O(n) complexity.
* The middle loop, nested within the outer loop, also has O(n) complexity.
* The inner loop, nested within the middle loop, also has O(n)complexity.

When loops are nested, their complexities multiply:

O(n)×O(n)×O(n)=O(n^3)

**Conclusion:**

The time complexity of the given program is:

O(n^3)

This cubic time complexity indicates that the number of operations grows proportionally to the cube of the input size nnn.