**C CODE TO ANALYSE COMPLEXITY OF QUICK SORT**

#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: Places the pivot in its correct position

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

int pivot = arr[high]; // Choosing the last element as pivot

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

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

if (arr[j] <= pivot) {

i++;

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

}

}

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

return (i + 1); // Return the partition index

}

// Quick Sort function

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

if (low < high) {

int pi = partition(arr, low, high); // Partitioning index

quickSort(arr, low, pi - 1); // Recursively sort the left part

quickSort(arr, pi + 1, high); // Recursively sort the right part

}

}

// Function to print the array

void printArray(int arr[], int size) {

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

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

}

printf("\n");

}

// Driver function to analyze time complexity

int main() {

int n;

printf("Enter the number of elements: ");

scanf("%d", &n);

int\* arr = (int\*)malloc(n \* sizeof(int));

if (arr == NULL) {

printf("Memory allocation failed!\n");

return 1;

}

// Input array elements

printf("Enter %d elements:\n", n);

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

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

}

printf("Unsorted Array: ");

printArray(arr, n);

// Measure execution time

clock\_t start, end;

start = clock(); // Start time

quickSort(arr, 0, n - 1);

end = clock(); // End time

printf("Sorted Array: ");

printArray(arr, n);

// Calculate the time taken for Quick Sort

double time\_taken = ((double)(end - start)) / CLOCKS\_PER\_SEC;

printf("Quick Sort executed in: %f seconds\n", time\_taken);

free(arr); // Free the allocated memory

return 0;

}