**Quicksort**

Design and implement C/C++ Program to sort a given set of n integer elements using Quick Sort method and compute its time complexity. Run the program for varied values of n> 5000 and record the time taken to sort. Plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.

#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 and return the pivot index

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

{

int pivot = arr[high]; // pivot

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

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

{

// If current element is smaller than or equal to pivot

if (arr[j] <= pivot)

{

i++; // increment index of smaller element

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

}

}

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

return (i + 1);

}

// Function to implement QuickSort

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

{

if (low < high)

{

// pi is partitioning index, arr[pi] 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);

}

}

int main()

{

int n;

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

scanf("%d", &n);

int arr[n];

// Generate random numbers for the array

srand(time(NULL));

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

{

arr[i] = rand() % 10000; // You can adjust the range of random numbers as needed

}

// Measure the time taken to sort the array

clock\_t start, end;

double cpu\_time\_used;

start = clock();

quickSort(arr, 0, n - 1);

end = clock();

cpu\_time\_used = ((double) (end - start)) / CLOCKS\_PER\_SEC;

printf("Sorted array:\n");

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

{

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

}

printf("\n");

printf("Time taken for sorting: %f seconds\n", cpu\_time\_used);

return 0;

}