**Assignment 2**

Name : Sri Sai Preethi Munnaluri

Class : CSAI-B

Roll No : 50

**Title:**

Write a program to implement Quick Sort to sort an array of integers in ascending order. Find out Time and space complexity.

**Objective:**

* To understand the mechanism of comparison-based sorting.
* To implement Bubble Sort in C++ or Java.
* To analyze time and space complexity of Quick Sort.

**Software Requirements:**

* Operating System: Windows/Linux
* Language: C++ or Java
* Compiler: g++/javac

**Hardware Requirements:**

* Processor: 2 GHz or above
* RAM: 4 GB or more
* Disk Space: Minimum 500 MB

**Theory:**

Quick sort is a highly efficient **divide-and-conquer** sorting algorithm. It works by selecting a **pivot element** from the array and partitioning the other elements into two sub-arrays: those less than the pivot and those greater than the pivot. The sub-arrays are then recursively sorted. This process continues until the entire array is sorted**.**

Algorithm:

**Step 1 −** Choose a pivot element from the array (commonly the first, last, or middle element). **Step 2 −** Rearrange the array such that all elements less than the pivot are placed before it, and all greater elements are placed after it (this is called partitioning).

**Step 3 −** Place the pivot in its correct position in the sorted array.

**Step 4 −** Recursively apply the same process to the left sub-array (elements smaller than the pivot) and right sub-array (elements greater than the pivot).

**Step 5 −** Continue until the base case is reached (sub-array size ≤ 1).

**Pseudocode of Quick sort:**

Start

Function quickSort(arr, low, high):

if low < high:

pi = partition(arr, low, high)

quickSort(arr, low, pi - 1) // Recursively sort left sub-array

quickSort(arr, pi + 1, high) // Recursively sort right sub-array

Function partition(arr, low, high):

pivot = arr[high] // Choose last element as pivot

i = low - 1

for j = low to high-1:

if arr[j] < pivot:

i = i + 1

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

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

return (i + 1)

End

**Time Complexity:**

| **Best Case** | O(n log n) |
| --- | --- |
| Average Case | O(n log n) |
| Worst Case | O(n²) |

**Space Complexity:**

Quick sort is an in-place sorting algorithm (no extra array is required). However, it uses the call stack due to recursion.

* Space Complexity = O(log n) for the recursion stack (best/average).
* In the worst case (highly unbalanced partitions), space can grow to O(n).

**CODE:**

public class QuickSort

{

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

    {

        int pivot = arr[high];

        int i = (low - 1);

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

        {

            if (arr[j] < pivot)

            {

                i++;

                int temp = arr[i];

                arr[i] = arr[j];

                arr[j] = temp;

            }

        }

        int temp = arr[i + 1];

        arr[i + 1] = arr[high];

        arr[high] = temp;

        return i + 1;

    }

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

    {

        if (low < high)

        {

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

            quickSort(arr, low, pi - 1);

            quickSort(arr, pi + 1, high);

        }

    }

    public static void main(String[] args)

    {

        java.util.Scanner sc = new java.util.Scanner(System.in);

        System.out.print("Enter the number of elements in the array:");

        int n = sc.nextInt();

        int[] arr = new int[n];

        System.out.println("Enter the elements of the array: ");

        for (int i = 0; i < n; i++)

        {

            arr[i] = sc.nextInt();

        }

        quickSort(arr, 0, n - 1);

        System.out.println("Sorted array:");

        for (int i = 0; i < arr.length; i++)

        {

            System.out.println(arr[i]);

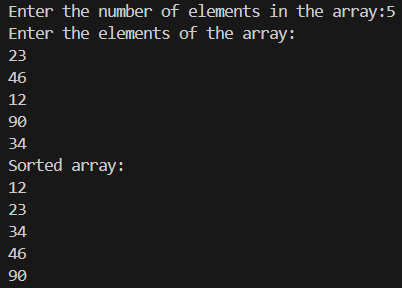
        }

        sc.close();

    }

}

**OUTPUT:**



**Time Complexity Analysis:**

* Best Case (balanced partitions): O(n log n)
* Average Case: O(n log n)
* Worst Case (already sorted or reverse sorted with bad pivot choice): O(n²)

**Space Complexity Analysis:**

* Quick Sort is in-place, meaning it does not need extra space for arrays.
* Uses O(log n) space for recursion stack in best/average cases.
* In worst case, recursive stack can grow to O(n).

**Conclusion:**

Quick sort is one of the fastest sorting algorithms for large datasets due to its **divide-and-conquer** approach and average-case **O(n log n)** time complexity.  
However, in the **worst case** (unbalanced partitions), its performance can degrade to **O(n²)**.  
It is widely used in practice and often outperforms other simple algorithms like **Bubble Sort** and **Insertion Sort**.